Digital Filters

Study the ImageAccess class

The class ImageAccess contains methods useful such as getRow, getNeighborhood, putColumn.

For example, the getNeighborhood method returns an n*n arrangement centered at a certain position (x,y).

ImageAccess also contains basic arithmetic operations:

Operations	**Syntax**	**Meaning**
Absolute value	image.abs();	Image <- abs(image)
Square root	image.sqrt();	Image <- sqrt(image)
Square	image.pow(2.0);	Image <- image ^ 2
Addition	image.add(image1, image2);	Image <- image1 + image2
Subtraction	image.subtract(image1, image2);	Image <- image1 - image2
Multiplication	image.multiply(image1, image2);	Image <- image1 * image2
Duplicate	image = image1.duplicate();	Image <- image1
Show the image	image.show("title of the window");	Show an image in a new window

1. Edge Detection Filter

The vertical edge detection mask is defined below:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

1.1 Understanding the Vertical Edge Detection Filter

Read and understand the code of a vertical edge detector in FilteringSession.java.

There are two versions:

- detectEdgeVertical_NonSeparable (): Non-separable version;

- detectEdgeVertical_Separable (): Separable version.

Apply this function to mit.tif and octagon images.tif selecting the right operation.

1.2 Writing the horizontal edge detector

Write a method that implements the separable version of the horizontal edge detector. The method name is detectEdgeHorizontal_Separable() in FilteringSession.java.

Apply this function to the mit.tif and octagon images.tif selecting the right operation and clicking "Run Student Solution".

1.3 Comparison

Compare the results of the two versions regarding compute time and image content. Do they differ or are they the same? Complete the report.doc. Apply this vertical and horizontal edge detector in africa.tif. Convert the results to 8-bit and insert it into the report.doc.

2. Moving averages (5x5)

The 5x5 moving average filter replaces one pixel with its average in a 5x5 centered window. There are several ways to implement a mobile averages filter; All give the same results, but some are more efficient than others. Here, we'll look at three versions of the implementation we call:

- Non-separable: The 5x5 window is run for each pixel of the image;
- Separable: The moving average routine is implemented in 1D. It is applied to all rows and columns;
- Recursive: Like the previous one, this method is also separable, but the routine computes 1D averaging using a more efficient recursive algorithm.

2.1 Writing the 5x5 non-breaking version

Write a method that implements the 5x5, non-breaking version of a

Moving average filter.

The name of the method is doMovingAverage5_NonSeparable ().

2.2 Writing the separable 5x5 version

Write a method that implements the separable version of a 5x5 moving average filter. The name of the method is doMovingAverage5_Separable ().

Create a 1D routine named doAverage5().

Pay attention to the implementation of reflective boundary conditions in the 1D routine.

2.3 Writing the separable 5x5 recursive version

Write a method that implements the 5x5, separable, recursive version of a moving average filter. The method name is doMovingAverage5_Recursive().

Create a 1D routine called doAverage5_Recursive() that implements a moving average, recursively.

Reflective boundary conditions are applied.

3. Smoothing operator applications

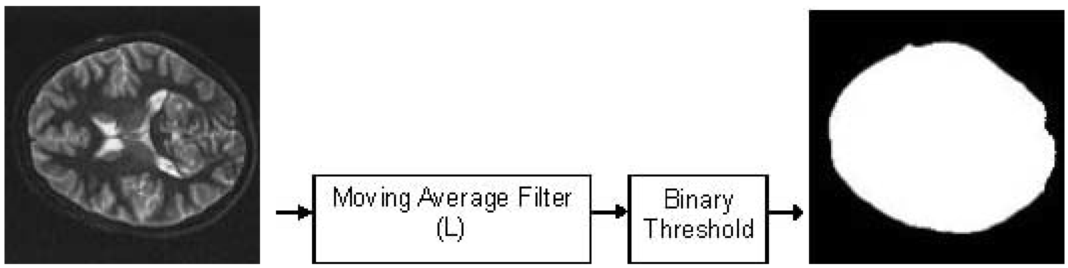
The 5x5 moving average filter replaces one pixel with its average in a 5x5 centered window. There are several ways to implement a moving average filter; All give the same results, but some are faster than others.

Here, we'll look at three versions of the implementation we call:

- Non-separable: The 5x5 window is traversed for each pixel of the image;
- Separable: The change-average routine is implemented in 1D. It is applied to all rows and all columns;
- Recursive: Like the previous one, this method is also separable, but the 1D routine computes the using a more efficient algorithm.

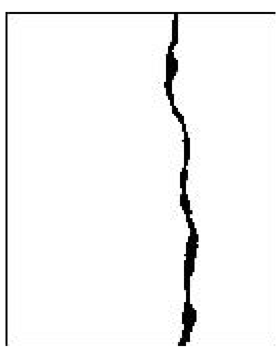
3.1 Segmenting

Using the diagram described below, segment the brain14.tif image into light and dark areas. Select an appropriate window length (L) for the moving average filter and choose an appropriate binary threshold (T) using the "Threshold" command from the Image/Adujst menu. Complete the report.doc.



3.2. Substantive deletion

Find a simple procedure to suppress the background that uses the smoother operator without a background image. The goal is to extract only the dendrite through thresholding. Apply the procedure to dendritis.tif and complete the report.doc.



4. Sobel Edge Detector

Type the doSobel() method that produces the following output:

$$c_s = \sqrt{(G_x\{f(x,y)\})^2 + \left(G_y\{f(x,y)\}\right)^2}$$

where:

- f(x,y) is the input image
- G_x, G_y are the vertical and horizontal gradients respectively, their masks are:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

If it is a separable filter, the filter becomes more efficient, so implement it in a separable way. Tip: Check out the image of Gx, Gy using the show("title") method of ImageAccess. Apply this function to mit.tif and octagon.tif images. Complete the report.doc.

5. Challenge: writing recursive moving average filter with variable window size.

Write a method that implements an LxL moving average filter that is faster than the separable version you previously developed (on average1D, and for a sufficiently large image, you should be able to access no more than two input pixels per output pixel).